CS 277: Database System Implementation

Notes 03: Disk Organization

Arthur Keller
Topics for today

• How to lay out data on disk
• How to move it to memory
What are the data items we want to store?

- a salary
- a name
- a date
- a picture

What we have available: Bytes

8 bits
To represent:

- Integer (short): 2 bytes
e.g., 35 is

\[
\begin{array}{c|c}
00000000 & 00100011 \\
\end{array}
\]

- Real, floating point
  \( n \) bits for mantissa, \( m \) for exponent....
To represent:

- Characters
  - various coding schemes suggested,
    most popular is ASCII

Example:
A: 1000001
a: 1100001
5: 0110101
LF: 0001010 (line feed)
To represent:

- **Boolean**
  
e.g., TRUE  
  $$\begin{array}{c}
  1111 \\
  1111 
  \end{array}$$
  
  FALSE  
  $$\begin{array}{c}
  0000 \\
  0000 
  \end{array}$$

- **Application specific**
  
e.g., RED → 1   
  GREEN → 3
  
  BLUE → 2   
  YELLOW → 4  ...

Can we use less than 1 byte/code?

Yes, but only if desperate...
To represent:

• Dates
  e.g.:  - Integer, # days since Jan 1, 1900
  - 8 characters, YYYYMMDD
  - 7 characters, YYYYDDDD
     (not YYMMDD! Why?)

• Time
  e.g.  - Integer, seconds since midnight
  - characters, HHMMSSFF
To represent:

- String of characters
  - Null terminated
    
    e.g.,
    
    cat
    
  - Length given
    e.g.,
    
    3 cat
    
- Fixed length
To represent:

- Bag of bits

<table>
<thead>
<tr>
<th>Length</th>
<th>Bits</th>
</tr>
</thead>
</table>
Key Point

• Fixed length items

• Variable length items
  - usually length given at beginning
Also

• Type of an item: Tells us how to interpret
  (plus size if fixed)
Overview

Data Items
  └── Records
    └── Blocks
      └── Files
        └── Memory
Record - Collection of related data items (called FIELDS)

E.g.: Employee record:
   name field,
   salary field,
   date-of-hire field, ...
Types of records:

• Main choices:
  – FIXED vs. VARIABLE FORMAT
  – FIXED vs. VARIABLE LENGTH
A **SCHEMA** (not record) contains following information

- # fields
- type of each field
- order in record
- meaning of each field
Example: fixed format and length

Employee record

(1) E#, 2 byte integer
(2) E.name, 10 char.
(3) Dept, 2 byte code

Records

<table>
<thead>
<tr>
<th>Employee</th>
<th>Dept</th>
</tr>
</thead>
<tbody>
<tr>
<td>55 smith</td>
<td>02</td>
</tr>
<tr>
<td>83 jones</td>
<td>01</td>
</tr>
</tbody>
</table>

Schema
Variable format

- Record itself contains format “Self Describing”
Example: variable format and length

Field name codes could also be strings, i.e. TAGS
Variable format useful for:

- “sparse” records
- repeating fields
- evolving formats

But may waste space...
• **EXAMPLE:** variable format record with repeating fields

Employee → one or more → children

| 3 | E_name: Fred | Child: Sally | Child: Tom |
Note: Repeating fields does not imply
- variable format, nor
- variable size

| John | Sailing | Chess | -- |

- Key is to allocate maximum number of repeating fields (if not used → null)
Many variants between fixed - variable format:

Ex. #1: Include record type in record length

| 5 | 27 | ... |

record type record length tells me what to expect (i.e. points to schema)
Record header - data at beginning that describes record

May contain:
- record type
- record length
- time stamp
- other stuff ...
Ex #2 of variant between FIXED/VAR format

- Hybrid format
  - one part is fixed, other variable

E.g.: All employees have E#, name, dept other fields vary.

| 25 | Smith | Toy | 2   | Hobby: chess | retired |

# of var fields
Also, many variations in internal organization of record

Just to show one:

```
<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>10</td>
<td>F1</td>
<td>5</td>
<td>F2</td>
<td>12</td>
<td>F3</td>
<td></td>
</tr>
</tbody>
</table>
```

- * length of field
- total size
- offsets
Question:
We have seen examples for
* Fixed format and length records
* Variable format and length records

(a) Does fixed format and variable length make sense?
(b) Does variable format and fixed length make sense?
Other interesting issues:

- Compression
  - within record - e.g. code selection
  - collection of records - e.g. find common patterns

- Encryption
Next: placing records into blocks

a file

assume fixed length blocks

blocks ...

assume a single file (for now)
Options for storing records in blocks:

(1) separating records
(2) spanned vs. unspanned
(3) mixed record types – clustering
(4) split records
(5) sequencing
(6) indirection
(1) Separating records

(a) no need to separate - fixed size recs.
(b) special marker
(c) give record lengths (or offsets)
   - within each record
   - in block header
(2) Spanned vs. Unspanned

- **Unspanned**: records must be within one block

  - **block 1**
    - R1
    - R2

  - **block 2**
    - R3
    - R4
    - R5

- **Spanned**

  - **block 1**
    - R1
    - R2
    - R3 (a)
    - R3 (b)

  - **block 2**
    - R4
    - R5
    - R6
    - R7 (a)

  ...
With spanned records:

need indication of partial record “pointer” to rest

need indication of continuation (+ from where?)
Spanned vs. unspanned:

• Unspanned is much simpler, but may waste space...

• Spanned essential if record size > block size
Example

$10^6$ records

each of size 2,050 bytes (fixed)

block size = 4096 bytes

```
<table>
<thead>
<tr>
<th>block 1</th>
<th>block 2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>R1</td>
<td>R2</td>
</tr>
<tr>
<td>2050 bytes</td>
<td>2050 bytes</td>
</tr>
<tr>
<td>wasted 2046</td>
<td>wasted 2046</td>
</tr>
</tbody>
</table>
```

- Total wasted = $2 \times 10^9$
- Utiliz. = 50%
- Total space = $4 \times 10^9$
(3) Mixed record types

- Mixed - records of different types (e.g. EMPLOYEE, DEPT) allowed in same block

E.g., a block:

| EMP | e1 | DEPT | d1 | DEPT | d2 |
Why do we want to mix?
Answer: CLUSTERING

Records that are frequently accessed together should be in the same block
Compromise:

No mixing, but keep related records in same cylinder ...
Example

Q1: select A#, C_NAME, C_CITY, ...
from DEPOSIT, CUSTOMER
where DEPOSIT.C_NAME = CUSTOMER.C_NAME

A block

CUSTOMER, NAME = SMITH
DEPOSIT, NAME = SMITH
DEPOSIT, NAME = SMITH
• If Q1 frequent, clustering good
• But if Q2 frequent
  Q2: SELECT *
      FROM CUSTOMER

CLUSTERING IS COUNTER PRODUCTIVE
(4) Split records

Typically for hybrid format

Fixed part in one block

Variable part in another block
Block with fixed recs.

```
R1 (a)
R2 (a)
```

---

Block with variable recs.

```
R1 (b)
R2 (b)
R2 (c)
```

This block also has fixed recs.
Question

What is difference between
- Split records
- Simply using two different record types?
(5) Sequencing

- Ordering records in file (and block) by some key value

Sequential file (⇒ sequenced)
Why sequencing?

Typically to make it possible to efficiently read records in order
(e.g., to do a merge-join — discussed later)
Sequencing Options

(a) Next record physically contiguous

(b) Linked
Sequencing Options

(c) Overflow area

<table>
<thead>
<tr>
<th>Records in sequence</th>
<th>header</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td></td>
</tr>
<tr>
<td>R2</td>
<td></td>
</tr>
<tr>
<td>R3</td>
<td></td>
</tr>
<tr>
<td>R4</td>
<td></td>
</tr>
<tr>
<td>R5</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Overflow area records</th>
</tr>
</thead>
<tbody>
<tr>
<td>R2.1</td>
</tr>
<tr>
<td>R1.3</td>
</tr>
<tr>
<td>R4.7</td>
</tr>
</tbody>
</table>
(6) Indirection

• How does one refer to records?

Many options:

Physical \[\leftrightarrow\] Indirect
Purely Physical

E.g., Record Address or ID = Device ID
                              Cylinder #
                              Track #
                              Block #
                              Offset in block

Block ID
Fully Indirect

E.g., Record ID is arbitrary bit string
Tradeoff

Flexibility ←→ Cost

to move records  of indirection

(for deletions, insertions)
Physical $\rightarrow$ Indirect $\leftarrow$

Many options in between ...
Ex #1 Indirection in block

A block:

- Header
- Free space
- R1
- R2
- R3
- R4
Block header - data at beginning that describes block

May contain:

- File ID (or RELATION or DB ID)
- This block ID
- Record directory
- Pointer to free space
- Type of block (e.g. contains recs type 4; is overflow, ...)
- Pointer to other blocks “like it”
- Timestamp ...
Ex. #2 Use logical block #’s understood by file system

REC ID → File ID
  Block #
  Record # or Offset

File ID, Block # → File System Map → Physical Block ID
File system map may be “Semi-physical”...

File F1: physical address of block 1  
table of bad blocks:  
\[
\begin{align*}
B57 &\rightarrow XXX \\
B107 &\rightarrow YYY
\end{align*}
\]

Rest can be computed via formula...
Num. Blocks: 20
Start Block: 1000
Block Size: 100
Bad Blocks:
   3 → 20,000
   7 → 15,000

Where is Block # 2?
Where is Block # 3?

File DEFINITION
Options for storing records in blocks

(1) Separating records
(2) Spanned vs. Unspanned
(3) Mixed record types - Clustering
(4) Split records
(5) Sequencing
(6) Indirection
Other Topics

(1) Insertion/Deletion
(2) Buffer Management
(3) Comparison of Schemes
Deletion

Block

Rx
Options:

(a) Immediately reclaim space
(b) Mark deleted
   – May need chain of deleted records (for re-use)
   – Need a way to mark:
     • special characters
     • delete field
     • in map
As usual, many tradeoffs...

- How expensive is it to move a valid record to free space for immediate reclaim?
- How much space is wasted?
  - e.g., deleted records, delete fields, free space chains,...
Concern with deletions

Dangling pointers

R1 → ?
Solution #1: Do not worry
Solution #2: Tombstones

E.g., Leave “MARK” in map or old location

- Physical IDs

```
A block

This space can never be re-used
```

This space can be re-used
Solution #2: Tombstones

E.g., Leave “MARK” in map or old location

- Logical IDs

<table>
<thead>
<tr>
<th>ID</th>
<th>LOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>7788</td>
<td>🏙️</td>
</tr>
</tbody>
</table>

Never reuse ID 7788 nor space in map...
Solution #3 (?):

- Place record ID within every record
- When you follow a pointer, check if it leads to correct record

Does this work???
If space reused, won’t new record have same ID?
Solution #4 (?):

• To point, use (pointer + hash) or (pointer + key)?

• What if record modified???
Insert

Easy case: records not in sequence
   → Insert new record at end of file or in deleted slot
   → If records are variable size, not as easy...
Hard case: records in sequence
   → If free space “close by”, not too bad...
   → Or use overflow idea...
Interesting problems:

- How much free space to leave in each block, track, cylinder?
- How often do I reorganize file + overflow?
Buffer Management

- DB features needed
- Why LRU may be bad
- Pinned blocks
- Forced output
- Double buffering
- Swizzling

Read
Textbook!
in Notes02
Swizziling

Memory

block 1

block 2

Rec A

Disk

block 1

block 2

Rec A
**One Option:**

<table>
<thead>
<tr>
<th>Translation Table</th>
<th>DB Addr</th>
<th>Mem Addr</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rec-A</td>
<td>Rec-A-inMem</td>
<td></td>
</tr>
</tbody>
</table>
Another Option:

In memory pointers - need “type” bit

\[
\begin{aligned}
\text{X} & \quad \text{to disk} \\
\text{M} & \quad \text{to memory}
\end{aligned}
\]
Swizziling

- Automatic
- On-demand
- No swizzling / program control
Comparison

- There are 10,000,000 ways to organize my data on disk...

Which is right for me?
Issues:

Flexibility \[\underline{\text{Space Utilization}}\]
\[\underline{\text{Complexity}}\] \[\underline{\text{Performance}}\]
To evaluate a given strategy, compute following parameters:

-> space used for expected data
-> expected time to
   - fetch record given key
   - fetch record with next key
   - insert record
   - append record
   - delete record
   - update record
   - read all file
   - reorganize file
Example

How would you design Megatron 3000 storage system? (for a relational DB, low end)
- Variable length records?
- Spanned?
- What data types?
- Fixed format?
- Record IDs?
- Sequencing?
- How to handle deletions?
Summary

- How to lay out data on disk

Data Items
  ↙
Records
  ↙
Blocks
  ↙
Files
  ↙
Memory
  ↙
DBMS
How to find a record quickly, given a key